okjji12

# github URL (optional):

|  |  |
| --- | --- |
|  | https://github.com/okjji12/Cloud\_Calculator |

# #개요

|  |  |
| --- | --- |
|  | **Topic** - Calculator application using Java socket API  **Language** - Java  **\* Program execution mechanism**  1. Server - Client Connection  2. Client sends formula to server  3. Determine if the formula received from the client is the proper formula format  4. Proper formula -> the correct answer of the formula to the client  Invalid formula -> Send information about error to the client  **\* Program Requirements**  1. Define a protocol for Command/Response (communication message formats)  2. The calculator should support the following four operations  - addition, subtraction, multiplication and division  3. The response from the server may be either answer for the expression or error message  4. The server can handle multiple clients at a time by Thread Pool & Runnable interface  5. The client attempts to connect by reading a text file containing the server's IP and port number information |

**Requirements**

* <요구조건 #1: 간략한 핵심 요구 조건 설명>

# REPORT

* 과제의 기본 요구 사항을 이곳에 원하는 만큼 작성 (e.g. 구조도, protocol 내용 상세 설명 등)
* Procotol에 대한 설명
* 소스코드는 마지막에 제공하는 양식을 이용하여 제출 (zip파일로도 제출)

# PRotocol

|  |  |
| --- | --- |
|  | **<Command>**  **ADD -** Execute add operation on two integers received from the client  **SUB -** Execute subtraction operation on two integers received from the client  **MUL -** Execute multiplication operation on two integers received from the client  **DIV -** Execute division operation on two integers received from the client  **<Response>**  **401 variable type**  - Error occurred when the expression contains non integer elements  **402 too many**  **-** Error occurred When there are more than three arguments in the expression  **403 div zero**  **-** Error occurred When the expression contains operation divided by zero  **404 too little**  **-** Error occurred When there are less than three arguments in the equation  **405 undef err**  **-** Errors other than defined errors occurred in the expression sent by client  **200 ok**  **-** When the calculation was completed successfully by determining that the correct expression has been sent by client |

# ArChITECTURE DIAGRAM

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

## SOURCE CODES (cloud\_calc\_server.java)

|  |
| --- |
| package calc\_s;  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.OutputStreamWriter;  import java.net.ServerSocket;  import java.net.Socket;  import java.util.StringTokenizer;  import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;  public class cloud\_calc\_server {  public static String answer; // static variable representing the result of the formula from client  // Method defining response status for client  public static String responseType(int resNo) {  String response = "";  if(resNo == 401) {  response = "401 variable type";  }else if(resNo == 402) {  response = "402 too many";  }else if (resNo == 403) {  response = "403 div zero";  }else if (resNo == 404) {  response = "404 too little";  }else if (resNo == 405) {  response = "405 undef err";  }else if(resNo == 200) {  response = "200 ok " + answer;  }  return response;  }    // Method determining whether an expression contains non - integer elements , associated with Error 401  public static boolean isInteger(String strValue) {  try {  Integer.parseInt(strValue);  return true;  } catch (NumberFormatException ex) {  return false;  }  }  // Method of determining the response type for the client and returning it in string form  public static String calc(String exp) {  StringTokenizer st = new StringTokenizer(exp, " ");  // Determine if the length of the formula is appropriate  if (st.countTokens() != 3) {  if (st.countTokens() < 3) {  return "Error 404";  } else if (st.countTokens() > 3) {  return "Error 402";  }  }  String[] array = exp.split(" ");  // The part that determines if it causes Error 401  if (isInteger(array[1]) == false || isInteger(array[2]) == false) {  return "Error 401";  }  String res = "";  // The process of calculating by splitting each element of the client's request format  String opcode = st.nextToken();  int op1 = Integer.parseInt(st.nextToken());  int op2 = Integer.parseInt(st.nextToken());  try {  switch (opcode) {  case "ADD":  answer = Integer.toString(op1 + op2);  break;  case "SUB":  answer = Integer.toString(op1 - op2);  break;  case "MUL":  answer = Integer.toString(op1 \* op2);  break;  case "DIV":  answer = Integer.toString(op1 / op2);  break;  default:  res = "Error 405";  return res;  }  } catch (ArithmeticException e) {  return "Error 403";  }  // Return the message that the calculation was successful unless there was a separate error  res = "200 OK";  return res;  }  // In a state of being able to receive requests from multiple clients by applying thread  private static class Calc\_Server\_Thread implements Runnable {  private Socket socket;  Calc\_Server\_Thread(Socket socket) {  this.socket = socket;  }  @Override  public void run() {  BufferedReader in = null;  BufferedWriter out = null;  try {  in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  while (true) {  String inputMessage = in.readLine();  if (inputMessage.equalsIgnoreCase("bye")) {  System.out.println("Client terminated the connection");  System.exit(0); // Shut down when server get "bye" from the client  }  System.out.println(inputMessage); // Outputs received messages to the screen  String res = calc(inputMessage); // res indicates the response status type  switch (res) {  case "Error 401":  res = responseType(401);  break;  case "Error 402":  res = responseType(402);  break;  case "Error 403":  res = responseType(403);  break;  case "Error 404":  res = responseType(404);  break;  case "Error 405":  res = responseType(405);  break;  case "200 OK":  res = responseType(200);  break;  default:  break;  }  out.write(res + "\n"); // Send calculation result string  out.flush();  }  } catch (IOException e) {  System.out.println(e.getMessage());  } finally {  try {  if (socket != null)  socket.close(); // Close the communication socket  } catch (IOException e) {  System.out.println("An error occurred while chatting with the client.");  }  }  }  }  public static void main(String[] args) throws Exception {  ServerSocket listener = null;  listener = new ServerSocket(9999); // Creating a Server Socket  System.out.println("Waiting for the connection.....");  ExecutorService pool = Executors.newFixedThreadPool(10);  try {  Socket sock = listener.accept(); // Waiting for connection requests from clients  System.out.println("Connected.");  pool.execute(new Calc\_Server\_Thread(sock));  } catch (Exception e) {  System.out.println(e.getMessage());  } finally {  try {  if (listener != null) {  listener.close(); // Close the socket of the server  }  } catch (Exception e2) {  // TODO: handle exception  }  }  }  } |

## SOURCE CODES (cloud\_calc\_client.java)

|  |
| --- |
| package calc\_c;  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileReader;  import java.io.IOException;  import java.io.InputStreamReader;  import java.io.OutputStreamWriter;  import java.net.Socket;  import java.util.Scanner;  import java.util.StringTokenizer;  public class cloud\_calc\_client {    // Method for determining response type from server  public static String resHandle(int errNo) {  String respon = "";  if (errNo == 401) {  respon = "Error - Variable type error";  } else if (errNo == 402) {  respon = "Error - too many arguments";  } else if (errNo == 403) {  respon = "Error - divided by zero";  } else if (errNo == 404) {  respon = "Error - too little arguments";  } else if (errNo == 405) {  respon = "Error - Undefined error";  }  return respon;  }  // Methods for defining request forms  public static String reqType(String oper) {  String request = "";  if (oper == "+") {  request = "ADD";  } else if (oper == "-") {  request = "SUB";  } else if (oper == "\*") {  request = "MUL";  } else if (oper == "/") {  request = "DIV";  }  return request;  }  public static void main(String[] args) {  BufferedReader in = null;  BufferedReader br = null;  BufferedWriter out = null;  Socket socket = null;  Scanner scanner = new Scanner(System.in);  // Default value in case the file does not exist  String ipAddress = "localhost";  String portNum = "9999";  try {  // Text file that contains the information about server IP address and port number  String fileName = "src/TextFile/server\_info.txt";  FileReader fr = new FileReader(fileName);  br = new BufferedReader(fr);  // Assigning an IP address and a port number by reading two lines of a text file  ipAddress = br.readLine();  portNum = br.readLine();  // Convert the data type of "portNum" to integer  socket = new Socket(ipAddress, Integer.parseInt(portNum));  fr.close();  br.close();  in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  while (true) {  System.out.print("Formula (enter in blank spaces, e.g. 24 + 42)>>"); // Input Guidelines  String outputMessage = scanner.nextLine(); // Read formula from keyboard  StringTokenizer st = new StringTokenizer(outputMessage, " ");  if (st.countTokens() == 3) {  String[] array = outputMessage.split(" ");  String opercd = array[1];  // Formatting messages sent to the server differently depending on the entered arithmetic operation  switch (opercd) {  case "+":  outputMessage = reqType("+") + " " + array[0] + " " + array[2];  break;  case "-":  outputMessage = reqType("-") + " " + array[0] + " " + array[2];  break;  case "\*":  outputMessage = reqType("\*") + " " + array[0] + " " + array[2];  break;  case "/":  outputMessage = reqType("/") + " " + array[0] + " " + array[2];  break;  default:  break;  }  } else if (outputMessage.equalsIgnoreCase("bye")) {  out.write(outputMessage + "\n"); // Send "bye" string  out.flush();  break; // If the user enters "bye", send it to the server and terminate the connection  }  out.write(outputMessage + "\n"); // Send formula string read from keyboard  out.flush();  String inputMessage = in.readLine(); // Receive calculation results from the server  String sub\_inputM[] = inputMessage.split(" ");  // Use the resHandle method to determine the meaning of the response sent by the server  switch (inputMessage) {  case "401 variable type":  inputMessage = resHandle(401);  break;  case "402 too many":  inputMessage = resHandle(402);  break;  case "403 div zero":  inputMessage = resHandle(403);  break;  case "404 too little":  inputMessage = resHandle(404);  break;  case "405 undef err":  inputMessage = resHandle(405);  break;  default:  inputMessage = sub\_inputM[2];  }  System.out.println("Result: " + inputMessage);  }  } catch (IOException e) {  System.out.println(e.getMessage());  } finally {  try {  scanner.close();  if (socket != null)  socket.close(); // Close the socket of the client  } catch (IOException e) {  System.out.println("An error occurred while chatting with the server.");  }  }  }  } |

// 위 내용 지우고 사용. source 코드 별로 위 양식을 복사하여 사용

## OUTPUT (Screen Shots)

|  |  |
| --- | --- |
| #1 | Result 1 (위는 server, 아래는 client) |
| 텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진  자동 생성된 설명 | |

|  |  |
| --- | --- |
| #2 | Result 2 (위는 server, 아래는 client) |
| 텍스트, 폰트, 소프트웨어, 스크린샷이(가) 표시된 사진  자동 생성된 설명텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진  자동 생성된 설명 | |

|  |  |
| --- | --- |
| #3 | Result 3 (위는 server, 아래는 client) |
| 텍스트, 폰트, 스크린샷이(가) 표시된 사진  자동 생성된 설명텍스트, 폰트, 소프트웨어, 웹 페이지이(가) 표시된 사진  자동 생성된 설명 | |

//(본인이 실행한 화면을 출력)  
//(최소 3개 이상 또는 필요한 만큼 더 생성하여 실행 결과를 잘 파악할 수 있도록 제시)   
// (Screen shot이 없는 경우 생략 가능)